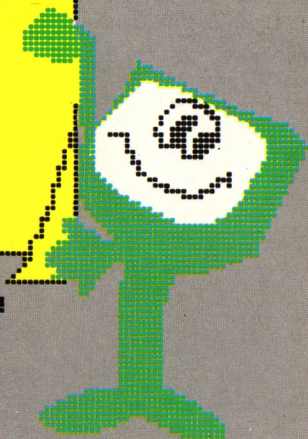
20 VIDEOLEZIONI DI BASIC
PER IMPARARE CON L'MSX

I computer del futuro
L'intelligenza artificiale

Il BASIC e la memorizzazione
dei programmi

Gli interrupt

Usare la ROM

BASE, VDP

Videoesercizi

Videogioco n° 20

VIDEOBASIC MSX
Pubblicazione quattordicinale
edita dal Gruppo Editoriale Jackson
Direttore Responsabile:
Giampietro Zanga
Direttore e Coordinatore
Editoriale: Roberto Pancaldi
Autore: Softidea -
Via Indipendenza 88-90 - Como
Redazione software:
Giuliano Cremonesi
Francesco Franceschini
Progetto grafico:
Studio Nuovidea - via Longhi, 16 - Milano
Impaginazione:
Moreno Confalone
Illustrazioni:
Cinzia Ferrari, Silvano Scolari
Fotografie:
Marcello Longhini
Distribuzione: SODIP
Via Zuretti, 12 - Milano
Fotocomposizione: Lineacomp S.r.l.
Via Rosellini, 12 - Milano
Stampa: Grafika '78
Via Trieste, 20 - Pioltello (MI)
Direzione e Redazione:
Via Rosellini, 12 - 20124 Milano
Tel. 02/6880951/5

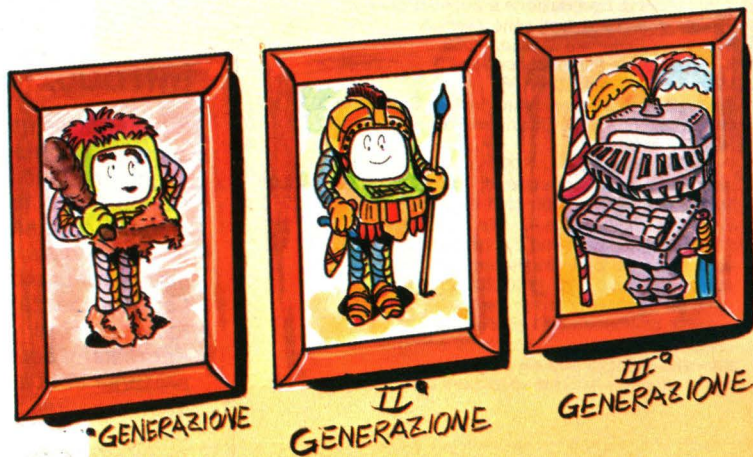
Autorizzazione alla pubblicazione Tribunale di Milano n° 422 del 22-9-1984
Spedizione in abbonamento postale Gruppo II/70 (autorizzazione della Direzione Provinciale delle PPTT di Milano).
Prezzo del fascicolo L. 8.000
Abbonamento comprensivo di 5 raccoglitori L. 165.000
I versamenti vanno indirizzati a: Gruppo Editoriale Jackson S.p.A. - Via Rosellini, 12 20124 Milano, mediante emissione di assegno bancario o cartolina vaglia oppure utilizzando il c.c.p. n° 11666203.
I numeri arretrati possono essere richiesti direttamente all'editore inviando L. 10.000 cdu. mediante assegno bancario o vaglia postale o francobolli.
Non vengono effettuate spedizioni contrassegno.

SOMMARIO

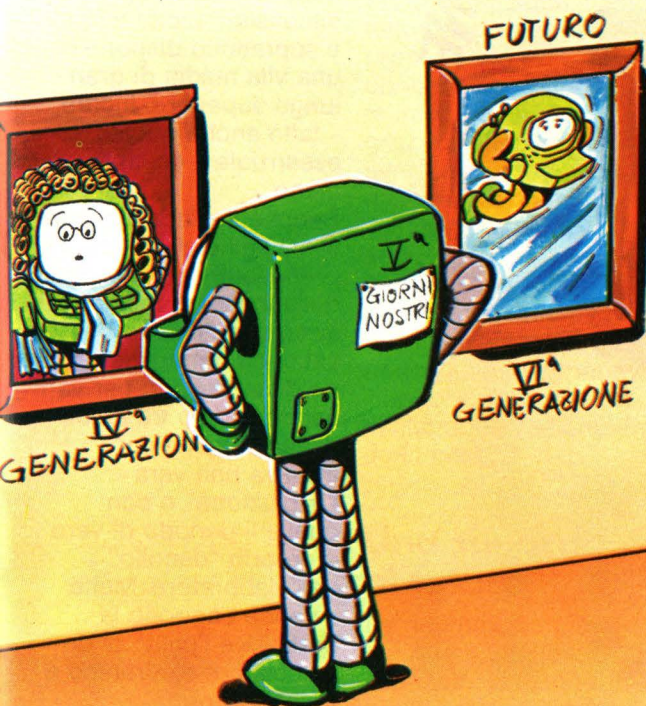
Introduzione

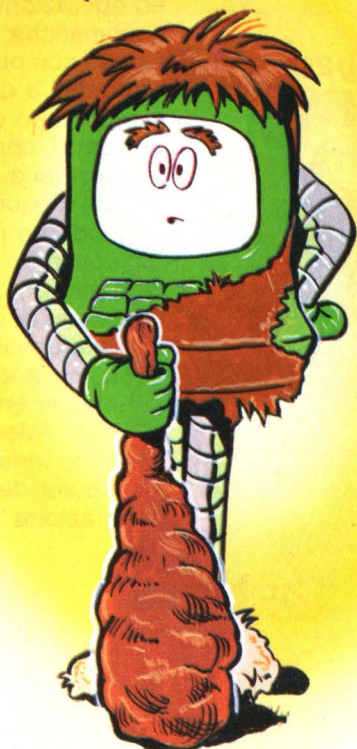
Il tuo MSX ha qualità e pregi impensabili sino a pochi anni fa; quello che i laboratori di ricerca ci stanno preparando, però va ben oltre: computer ultraveloci, capaci di elaborare più informazioni in parallelo, ma soprattutto computer "intelligenti". La fantascienza, insomma, non è più tanto futuribile.
Per restare, comunque, coi piedi nel presente dobbiamo perfezionare quanto appreso e apprendere del nuovo.
Ecco allora come risparmiare tempo e memoria, i puntatori, gli interrupt, e ancora linguaggio macchina.
Morale. Se il computer non ragiona e parla ancora come un uomo, occorre saper ragionare e parlare come il computer.

I computer del futuro

Per quanto molte persone ritengano tuttora il contrario, i computer non lavorano assolutamente per magia: ormai sappiamo benissimo che tutti i risultati ottenibili mediante un elaboratore elettronico non sono altro che il prodotto di un complesso e velocissimo insieme di operazioni elementari, svolte all'interno dei circuiti e delle memorie della macchina. Ciò che ad alcuni può ancora sembrare fantastico, è in realtà un preciso e logico sviluppo di un settore tecnologico che poggia su solide e rigorose basi teoriche e scientifiche.
Soltanto fino a pochi decenni fa nessuno avrebbe tuttavia potuto pronosticare la nascita e soprattutto lo sviluppo di una tecnologia così rivoluzionaria come quella elettronica: non esisteva infatti alcun presupposto che lasciasse intravvedere un avvenire così carico di sviluppi. Adesso che viviamo in pieno nella cosiddetta "era

elettronica" abbiamo invece sufficienti "orizzonti" per poter immaginare quali saranno i probabili progressi nel settore dei computer.
Le strade che in questo momento vengono battute dagli studiosi di tutto il mondo meritano comunque di essere descritte e analizzate.

I calcolatori del passato

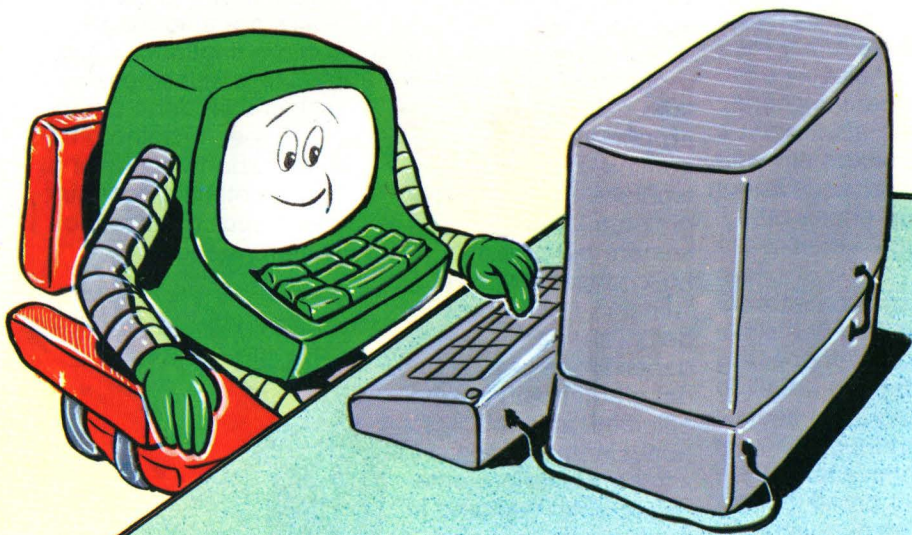
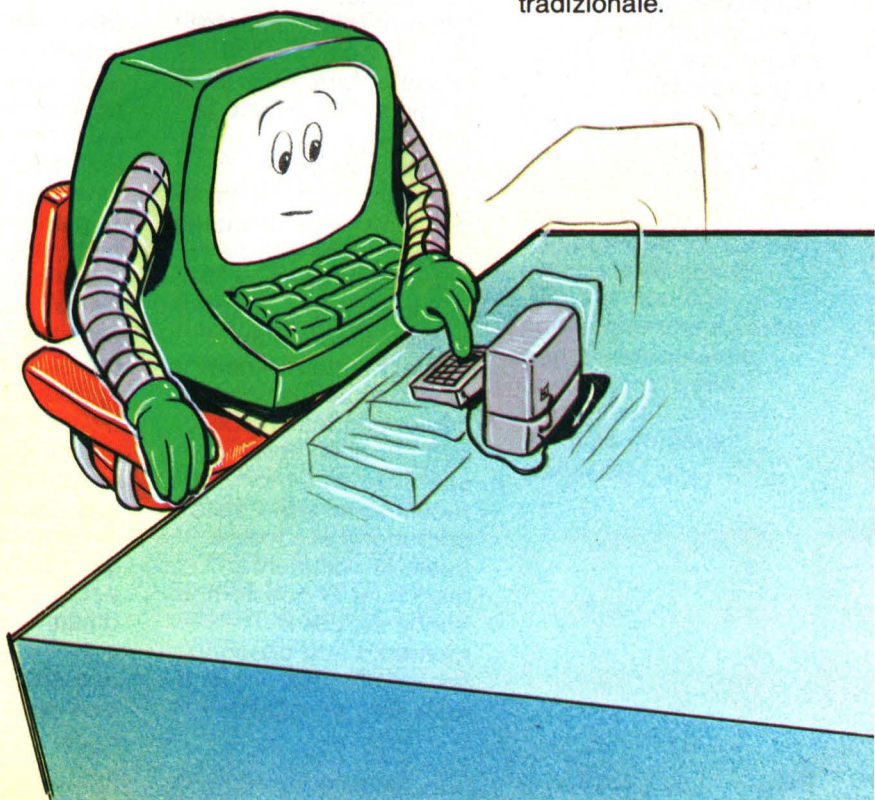
Prima di pensare al futuro diamo innanzitutto un'occhiata al passato: il detto "preparati al futuro guardando nel passato" è più che mai appropriato per chi, come noi, desidera fondare in maniera concreta le proprie ipotesi. Trascurando gli studi e le teorie logiche che hanno consentito la nascita e l'evoluzione del calcolo automatico, andiamo all'inizio degli anni '50, quando ancora nel campo elettronico imperavano le valvole e i diodi. La guerra era finita da poco e i calcolatori erano appena agli albori: in quel periodo cominciarono comunque ad apparire le prime macchine elettroniche capaci di eseguire automaticamente calcoli ed operazioni matematiche.
Questi computer raffrontati a quelli di oggi sembravano dei dinosauri, con dimensioni a dir poco enormi (l'equivalente di un moderno personal occupava un intero laboratorio), composti da chilometri di cavi e migliaia di valvole. In più consumavano notevoli quantità di energia elettrica. Adesso vengono indicati come "computer della prima generazione".

HARDWARE

Le macchine di questo tipo erano principalmente a disposizione dei grossi centri di ricerca militari od universitari e di grandi industrie: l'hardware era infatti troppo ingombrante e soggetto a guasti per garantire immediati sviluppi o applicazioni commerciali. Anche per il software le cose non andavano molto meglio: linguaggio macchina e linguaggi simbolici elementari di tipo assemblatore ponevano grossi limiti di utilizzo.
Però, pian piano gli elaboratori cominciarono comunque a diffondersi: occorreva però renderli più affidabili e meno ingombranti. Il fatto decisivo fu la scoperta del transistor. In un computer un transistor assolve la stessa funzione di una valvola: il transistor consuma però molto meno energia delle valvole, è di dimensioni molto minori e soprattutto dispone di una vita media di gran lunga superiore. Inoltre, - fatto anche questo essenziale - costano meno.
Benché sostanzialmente simili agli elaboratori precedenti sotto l'aspetto della logica, questi nuovi sistemi (detti della seconda generazione) se ne distaccavano per vari aspetti, primo tra tutti il dimensionamento.
Iniziava una vera "rivoluzione" e con quella il periodo di vero e proprio "decollo" dell'elaboratore. Molte aziende capirono la praticità e l'utilità di un calcolatore elettronico e lo installarono

richiedendo nello stesso tempo una maggiore potenza e ancora più elevata velocità di elaborazione. Nuove possibilità di elaborazione apparvero con l'introduzione delle memorie a nucleo magnetico, allargando ulteriormente le applicazioni e i possibili sviluppi.
Il più grosso passo avanti venne comunque fatto con la creazione dei circuiti integrati o chip: questa nuova tecnologia generò gli elaboratori della "terza generazione". I circuiti integrati introdussero nuovi miglioramenti, miniaturizzando e raffinando i componenti della seconda generazione. Essi inoltre costavano ancora meno delle "vecchie" piastre a transistor.
Contemporaneamente agli sviluppi dell'hardware, anche la programmazione aveva fatto passi da gigante, proponendo in continuazione nuove tecniche e applicazioni, grazie a linguaggi sempre più potenti e nello stesso tempo più "umani". I computer della "quarta generazione" sono quelli dei giorni nostri: piccoli, efficienti, affidabili, ma - nonostante le apparenze - ancora ulteriormente migliorabili. Vediamo in quali modi.

Uno sguardo al futuro

La ricerca scientifica è già a uno stadio avanzato: si stanno infatti battendo molte strade, sia "hardware" che "software". Le principali aree di studio dal punto di vista costruttivo riguardano soprattutto la superconduttività e l'elaborazione parallela, mentre l'informatica vera e propria punta tutte le proprie speranze verso quel settore di indagine, estremamente stimolante, che prende il nome di "intelligenza artificiale". Al solito, lo scopo è quello di realizzare elaboratori sempre migliori, sempre più versatili, sempre più utili; che lavorino più in fretta, memorizzino più informazioni, richiedano meno potenza, occupino meno spazio e costino sempre meno.

I computer a superconduttori

In realtà, i computer a superconduttori esistono già da qualche tempo: tuttavia le possibilità di sviluppo che essi sembrano in grado di offrire li pongono in un settore che appartiene più al domani che all'oggi.
I computer attualmente costruiti hanno raggiunto velocità di elaborazione talmente elevate da poter essere confrontate alle velocità con cui gli elettroni si muovono nei circuiti elettronici. È chiaro che se gli spostamenti degli elettroni all'interno dell'elaboratore (ricordiamoci che il flusso degli elettroni all'interno dei circuiti costituisce la corrente elettrica) sono più lenti delle possibilità di calcolo dell'unità centrale, il tempo di esecuzione è costretto a subire un rallentamento.
In altre parole, le informazioni impiegano un certo tempo (per quanto piccolissimo) per

andare da un punto all'altro dei circuiti elettronici: se questo tempo è superiore a quello della velocità di calcolo della CPU, questa è per forza di cose costretta a "rallentare", con ovvie conseguenze sulla velocità di lavoro dell'intero sistema. Questo problema, a prima vista apparentemente insolubile (non esiste infatti alcuna possibilità di "accelerare" il moto degli elettroni nei conduttori, essendo quest'ultimo una caratteristica specifica dei materiali), viene allora risolto ricorrendo a particolari leghe di metalli conduttori, che godono della singolare proprietà di opporre - a temperature estremamente basse (più di 100 gradi al di sotto dello zero!) - una resistenza al moto degli elettroni di gran lunga più bassa di quella che offrono alle normali temperature ambientali. Il problema viene quindi brillantemente risolto: le velocità di elaborazione dei computer a superconduttori (superconduttività è il nome del processo fisico che abbiamo appena visto) raggiungono infatti valori assolutamente impensabili nei comuni computer a tecnologia tradizionale.

I computer paralleli

L'idea che sta alla base della tecnica dei computer paralleli è di una semplicità quasi disarmante: anziché usare una sola unità centrale, nei computer paralleli si utilizzano più unità centrali, che lavorano in modo simultaneo o, come si usa dire più comunemente, in parallelo. Le possibilità che si propongono sono estremamente allettanti: teoricamente basta aggiungere altre CPU e le potenzialità di elaborazione di qualsiasi computer diventano praticamente senza limiti.

Naturalmente, come nella maggior parte dei progetti che a parole sembrano "semplici", gli spazi di azione che separano il dire dal fare appaiono estremamente impegnativi.
I problemi principali dei computer paralleli non risiedono infatti unicamente nelle pure e semplici disposizioni circuitali (tutt'altro che facili da risolvere), ma anche (e soprattutto) nelle modalità di programmazione che queste macchine richiedono.
Occorre infatti disporre di un linguaggio che riesca a "sincronizzare" le operazioni di tutte le CPU, evitando interferenze ed accavallamenti reciproci, con le ovvie (e deleterie) conseguenze che ne potrebbero derivare. A tutt'oggi un linguaggio di questo genere non esiste: i tentativi finora sperimentati lasciano comunque intravvedere ben più di semplici speranze. In tempi recentissimi sono state comunque prodotte e poste in commercio macchine di questo tipo, anche se, per il momento non possono essere sfruttate al pieno delle loro possibilità.

Intelligenza artificiale

L'intelligenza artificiale è senza alcun dubbio il settore che nello sviluppo della scienza dei computer avrà la maggiore influenza sul nostro modo di vivere nei prossimi decenni.
Per generazioni gli scrittori di fantascienza hanno pronosticato l'evoluzione di macchine più o meno intelligenti, capaci di assolvere molte delle funzioni eseguibili soltanto dagli esseri umani. Con ogni probabilità le vicende di androidi e umanoidi resteranno però di esclusivo dominio della lettura fantastica e avveniristica.
Al giorno d'oggi si pensa piuttosto alle macchine "intelligenti" come a sistemi capaci di prendere determinate decisioni al momento più opportuno.
L'esatta definizione dell'intelligenza di una macchina è un argomento in continua evoluzione: per quanto gli esperti si accaniscano in continui dibattiti su questo tema, si può comunque accettare come definizione standard

quella che venne proposta nei "lontani" anni '40 da un vero e proprio pioniere dell'informatica: Alan Turing. Piuttosto che elencare una serie di criteri da soddisfare per classificare un computer come intelligente, egli si limitò a dare una opinione ben più pratica del problema.
Turing affermò che se una persona non è in grado di distinguere se certe risposte le arrivano da una macchina o da un'altra persona, allora la macchina che ha eventualmente elaborato quelle risposte è da classificarsi come intelligente. Tale prova costituisce la base del famoso "Test di Turing", nel quale un operatore umano deve sostenere - attraverso una tastiera e un terminale - una certa conversazione, cercando di costringere l'interlocutore a svelare la propria identità di uomo o di macchina.
I centri di ricerca di tutto il mondo stanno portando avanti studi e indagini sulla intelligenza artificiale, e sembra che anche in questo caso sia solo questione di anni per arrivare a risultati effettivi. Il Giappone ha già anticipato tutti, annunciando che il suo elaboratore della "quinta generazione" vedrà la luce al massimo entro il 1995.
L'intelligenza artificiale ha già fatto il suo ingresso in molti settori, come per esempio quello dei "sistemi esperti": con questo nome si intendono quegli elaboratori specializzati in grado di eseguire un certo compito altrettanto bene (e in alcuni casi anche meglio) degli esperti umani. Un tipico esempio di utilizzo di questa tecnologia lo possiamo vedere tutti i giorni guardando le previsioni del tempo, elaborate quotidianamente appunto mediante l'ausilio di sistemi esperti. Anche nel campo delle diagnosi mediche sembra si stiano facendo passi da gigante: l'ipotesi del computer-dottore non è quindi troppo azzardata.
La maggior parte degli studi sull'intelligenza artificiale vengono condotti utilizzando particolari linguaggi, creati appositamente per questo scopo (in genere LISP e PROLOG). Poiché questi ultimi richiedono potenze di calcolo ben superiori a quelle offerte dai piccoli computer, nelle nostre case l'intelligenza artificiale entrerà tra pochi anni grazie alla telematica che consentirà di collegare l'home computer ad un grande sistema intelligente.
Ciò non significa comunque che il campo non sia affrontabile - anche se a livello hobbystico - con un normale personal computer.

Risparmiare tempo e memoria

Nella programmazione, come in molti altri settori del lavoro umano, è stata costruita una scala di valori, che classifica le varie tecniche in "buono", "non molto buono", "pessimo". Quando hai cominciato a usare il tuo MSX essere in grado di scrivere un programma funzionante era già di per sé un valido risultato. Adesso che sei diventato molto più padrone della situazione, e disponi di conoscenze e capacità che inizialmente non avevi, è giunto il momento di affrontare un discorso che - a prima vista - ti potrebbe sembrare poco rilevante, ma che in realtà è importantissimo per migliorare la tua tecnica di programmazione.
Vogliamo infatti vedere come migliorare ed aumentare la velocità di esecuzione dei tuoi programmi BASIC, senza per questo influenzarne la qualità, la leggibilità e l'occupazione di memoria. Non sempre velocità di esecuzione e risparmio di memoria sono conciliabili: esiste comunque un certo numero di "trucchetti", che di volta in volta possono tornare utili.
Vediamone quindi alcuni:

- **Istruzioni multiple.**
Il porre più istruzioni in una stessa linea aiuta a minimizzare la lunghezza del programma, risparmiando sui numeri di linea e di conseguenza sull'occupazione di memoria. Lo svantaggio e la limitazione principale di questa tecnica risiede però nella scarsa leggibilità e nella difficile modificabilità delle varie righe. È quindi meglio non abusare troppo con questa pratica.
- **Variabili.**
Le variabili dovrebbero essere chiamate con i nomi più brevi possibili. Questo aiuta a risparmiare memoria ed accelera il lavoro dell'interprete BASIC. Anche le costanti (sia numeriche che alfanumeriche) - se utilizzate di frequente - conviene assegnarle a delle variabili. Per esempio, anziché scrivere:

```
10 POKE 15143,12:POKE 15143,70:POKE 15143,20
```

conviene fare:

```
10 LET A=15143:POKE A,12:POKE A,70:POKE A,20
```

L'interprete BASIC dovrà in questo modo convertire una sola volta il numero 15143 in un formato comprensibile alla CPU, riducendo oltretutto anche lo spazio occupato in memoria.

• REM.
Bisogna limitare al massimo l'uso dei commenti nei programmi. Questa indicazione sembra in apparente contraddizione con quanto avevamo detto finora, e cioè che la documentazione dei programmi dovrebbe essere la più abbondante possibile. I programmatori più esperti risolvono allora il problema conservando due copie dello stesso programma: la prima, commentata in tutti i suoi aspetti, serve per capire il funzionamento del programma e per apportarvi eventuali modifiche; la seconda, che dovrà girare nel computer, viene privata di qualsiasi commento, in modo da evitare "inutili" sprechi di memoria.

• **GOTO.**
Ogni volta che l'interprete deve eseguire un salto la sequenza delle istruzioni subisce un improvviso cambiamento, spostandosi a un altro punto del programma. L'entità di questo cambiamento viene definita specificando nell'istruzione GOTO il numero di linea a cui saltare. L'interprete BASIC non dispone però di tecniche di ricerca particolari per individuare tale numero di linea ed esegue quindi una lenta ricerca sequenziale a partire dall'inizio del programma. Il tempo necessario per eseguire una istruzione di salto dipende dunque dalla lunghezza del testo e dalla distanza della linea indicata nel GOTO dall'inizio del testo. Quando si desidera accelerare al massimo la velocità di esecuzione è pertanto necessario valutare attentamente questo fatto, cercando di limitare al massimo la presenza di GOTO (avvantaggiando anche la leggibilità del programma) o - se proprio non se ne può fare a meno - avvicinando al massimo le istruzioni a cui saltare all'inizio del programma.

• **GOSUB.**
L'utilizzo delle subroutine consente un notevole risparmio di memoria, dal momento che permette di evitare la ripetuta scrittura di gruppi di istruzioni. Per i comandi GOSUB valgono tuttavia le stesse considerazioni fatte per i GOTO; la cosa migliore (e questo è di semplice applicazione) è allora quella di porre tutte le subroutine - all'inizio dei programmi, anziché alla fine come siamo abituati - dando la precedenza (cioè scrivendole prima) a quelle che il programma utilizzerà più frequentemente.
La raccomandazione più importante è comunque sempre la stessa e cioè che qualsiasi programma può diventare notevolmente più corto e veloce, se viene strutturato con una buona logica.

Cosa sono i puntatori

Un puntatore è una porzione della memoria del computer (di solito molto piccola: una o due locazioni) il cui contenuto è costituito da indirizzi utili al funzionamento del sistema. Ci spieghiamo meglio con un esempio. Quando accendi il tuo Spectrum, l'interprete BASIC deve mettersi immediatamente nella condizione di accettare le linee di programma che vorresti battere. Per fare questo è chiaramente necessario che l'interprete conosca la zona di memoria in cui dovrà immagazzinare le varie istruzioni; l'indirizzo di tale zona sarà allora contenuto in un apposito puntatore, letto dall'interprete durante la routine di accensione (o inizializzazione) del sistema. In ogni computer esistono numerosi puntatori, ciascuno dei quali con una specifica funzione. La loro principale utilità risiede nel fatto che

grazie ad essi è possibile riferirsi con minimo sforzo a particolari aree della memoria, consentendo quindi, con estrema facilità, eventuali modifiche o aggiornamenti.

La variabile Base

Come abbiamo già visto, tutte le informazioni relative alla pagina video, nei sistemi MSX, sono memorizzate in una particolare memoria RAM, denominata VRAM. Con il BASIC è possibile accedere alla VRAM mediante le istruzioni VPOKE e VPEEK; l'organizzazione della VRAM, però, non è assegnata a priori, e può essere modificata di volta in volta.
Per ovviare a questo inconveniente, che renderebbe impossibile scrivere programmi sicuramente funzionanti in ogni condizione, il BASIC mette a disposizione la variabile di sistema "BASE".
BASE è una variabile con indice, ovvero un vettore, che nella versione base MSX ha 20 componenti.
Ogni componente di base è un intero, e indica l'indirizzo di inizio, nella VRAM, di una particolare area della memoria video.
Per il resto, BASE si comporta esattamente come ogni altro vettore; si possono leggere i suoi componenti e si può assegnare loro dei valori.
Vediamo ora il significato di ciascuno dei componenti di "BASE", e qualche esempio del loro utilizzo.

Per il MODO 0:

BASE(0): indirizzo di inizio dell'area di memoria in cui viene memorizzato il testo da visualizzare.

BASE(1): inutilizzato.

BASE(2): inizio dell'area in cui sono memorizzati gli schemi dei caratteri.

BASE(3), (4): inutilizzati.

Per il MODO 1:

BASE(5): testo visualizzato.

BASE(6): colore dei caratteri.

BASE(7): forma dei caratteri.

BASE(8): attributi sprite.

BASE(9): forma gli sprite.

Per il MODO 2:

BASE(10):organizzazione dei gruppi di pixel 8x8.

BASE(11): colore dei pixel.

BASE(12): stato dei pixel.

BASE(13): attributi sprite.

BASE(14): forma sprite.

Per il MODO 3:

BASE(15): organizzazione dei gruppi di pixel 2x2.

BASE(16): inutilizzato.

BASE(17): stato dei pixel.

BASE(18): attributi sprite.

BASE(19): forma sprite.

In alcune applicazioni avanzate può essere utile cambiare i valori di base, ad esempio per memorizzare più pagine di testo, o per spostare rapidamente delle figure in alta risoluzione.
Comunque l'uso di BASE diventa indispensabile quando si desidera scrivere nella VIDEORAM, come ad esempio si fa per cambiare il set dei caratteri, per essere sicuri che la zona a cui si accede è veramente quella che interessa.
Immaginiamo di volere utilizzare un set di caratteri alternativo in modo schermo 1: non dobbiamo fare altro che modificare l'area di VRAM contenente le informazioni relative alla forma dei caratteri, inservendovi la definizione del nuovo set.
In generale tale area inizia all'indirizzo 0, e quindi potremmo essere tentati di scrivere un programma che modifichi la VRAM da 0 in avanti.
Un tale programma potrebbe, in qualche caso, non funzionare.
Un programma precedente, infatti, potrebbe avere modificato l'organizzazione della VRAM, spostando l'area di definizione dei caratteri ad un altro indirizzo e ponendo al suo posto ad esempio, l'area di definizione degli sprite.
In questo caso il nostro programma, invece di cambiare il set di caratteri, modificherebbe gli sprite!
Il programma funzionerà certamente se modificherà l'area che inizia all'indirizzo BASE(7), qualunque esso sia.
Vediamo qualche esempio:

```
PRINT BASE(14)
```

Visualizza l'indirizzo di inizio dell'area dedicata alla definizione della forma degli sprite nel MODO SCHERMO 2.

```
VPOKE BASE(0), 65
```

Poni il numero 65 nella prima posizione dell'area che contiene il testo per il modo SCHERMO 0.
Se ci si trova in MODO 0, si vedrà comparire, nell'angolo in alto a sinistra, una "A" (codice 65).

```
BASE(0)=&H800
```

Sposta l'area testo del MODO SCHERMO 0 all'indirizzo &H800.
Va notato che il sistema continuerà ad inviare i caratteri visualizzati al vecchio indirizzo finché non si eseguirà una istruzione SCREEN.

La funzione VDP

Nei sistemi MSX, tutte le funzioni relative alla visualizzazione delle immagini sullo schermo video sono svolte da un unico circuito integrato detto VDP (VIDEO DISPLAY PROCESSOR, O PROCESSORE VIDEO).

La sigla che identifica commercialmente il VDP MSX è TMS9929.
Come nel caso del PSG, anche il funzionamento del VDP è controllato dal contenuto di alcuni registri.
Il TMS9929 possiede 9 registri, di cui 8 a sola scrittura, e uno a sola lettura.
Quando scrive un nuovo valore nei registri a sola scrittura, il sistema MSX annota i dati scritti nella sua area di lavoro, in modo da conoscere il loro contenuto anche se non è più possibile rileggerlo.
I registri sono identificati dai numeri da 0 a 8; il registro 8 è quello a sola lettura.
Il contenuto dei registri del VDP viene analizzato e alterato da molte istruzioni Basic, come SCREEN, COLOR, BASE, ON SPRITE GOSUB, eccetera.
Il BASIC, però offre una possibilità in più per accedere ai registri del VDP con la funzione VDP.
VDP(N) rappresenta il registro N, e può essere trattato come una variabile. Naturalmente a VDP(8) non si può assegnare un valore.
In teoria la potenza del Basic MSX non dovrebbe far sentire al programmatore l'esigenza di utilizzare VDP; per ogni necessità esiste una adeguata istruzione Basic.
In pratica, però, manipolando opportunamente i registri si possono ottenere risultati difficili da raggiungere solo con il Basic.
Vediamo qualche esempio:

```
VDP(1)=VDP(1)AND191
```

In questo modo di pone a 0 il bit 6 del registro 1, in modo da nascondere lo schermo e fare apparire il solo colore dello sfondo.

```
VDP(1)=VDP(1)OR 64
```

Riporta lo schermo in condizioni normali. La sequenza delle istruzioni viste può servire ad oscurare lo schermo durante un draw, facendo poi apparire la figura tracciata all'improvviso, come se venisse dal nulla.

```
VDP(1)=VDP(1) OR 2
```

Poni ad 1 il bit del registro 1, per selezionare la misura ingrandita degli sprite. Lo stesso si potrebbe fare anche con SCREEN, 1 ; in questo modo, però, la defìnzione degli sprite verrebbe cancellata. Come hai visto, l'uso di VDP può dare risultati molto interessanti; attento, però, perché un errore potrebbe metterti in difficoltà, rendendo invisibili le operazioni, che stai compiendo. In ogni caso, potrai riportare le cose a posto resettando il computer, oppure spegnendolo e riaccendendolo.

Gli interrupt

La CPU usa l'area di stack per numerose operazioni, tra le quali anche la gestione degli interrupt. La comunicazione tra la CPU e le periferiche può avvenire con due tecniche diverse. La

prima prende il nome di "polling" (interrogazione ciclica): la CPU interroga ciclicamente, secondo un ordine prestabilito, i dispositivi esterni, usando un programma che legge lo stato degli stessi. Quando uno dei dispositivi è pronto a trasmettere o a ricevere un dato la CPU manda il programma necessario all'operazione richiesta. La priorità tra i diversi dispositivi è determinata dall'ordine con cui questi ultimi vengono interrogati.
Questa tecnica presenta un solo vantaggio, e cioè di essere realizzata completamente via software, e quindi di non richiedere circuiteria addizionale per le comunicazioni. Presenta però diversi svantaggi, che in alcuni casi la rendono addirittura impraticabile: la CPU è praticamente occupata per la maggior parte del tempo a interrogare dispositivi, mentre solo una minima parte di questo è impiegata per la comunicazione vera e propria. Inoltre il tempo di risposta della CPU alle richieste dei dispositivi può essere in certi casi troppo lungo: se ad esempio i dispositivi sono

parecchi, e uno di essi richiede un'operazione di I/O immediatamente dopo essere stato interrogato, deve aspettare parecchio prima di essere servito; in certi casi questo può comportare la perdita di dati.

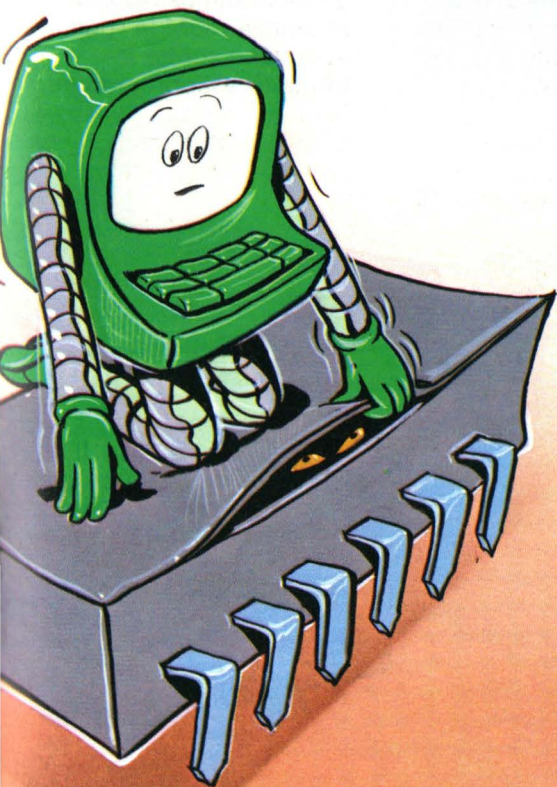
La seconda tecnica, che elimina questi svantaggi, è quella degli interrupt. Sono gli stessi dispositivi a segnalare alla CPU la richiesta di un'operazione di I/O, inviando un segnale, (chiamato appunto interrupt, o interruzione) per richiedere alla CPU di interrompere il programma che sta eseguendo, per effettuare l'operazione di I/O. Il servizio di interrupt avviene con un salto a una routine, che prende il nome di routine di servizio dell'interrupt, la quale provvede ad eseguire l'operazione richiesta. Questa tecnica presenta

molte analogie con l'esecuzione della subroutine, con la differenza che l'esecuzione della routine di servizio avviene in momenti non prevedibili dal programma, che dipendono dalle esigenze dei dispositivi esterni.

È chiaro che la tecnica degli interrupt elimina gli svantaggi cui si era prima accennato. Il tempo di risposta è infatti limitato solo dalla velocità della CPU per trasferire il controllo da una zona a un'altra della memoria; inoltre l'unità centrale può dedicarsi ad altri programmi, utilizzando in modo più efficiente il suo tempo.
La differenza tra polling e interrupt è la stessa che ci sarebbe tra aprire la porta di casa ad intervalli di tempo prefissati, per vedere se c'è qualcuno, e il rispondere al trillo del campanello. La perdita di tempo nel primo caso è evidente, come lo è la scomodità del servizio per chi, desiderando comunicare, deve attendere il successivo controllo per poterlo fare. Dobbiamo però dire che per la gestione delle interruzioni, oltre al dispositivo (il campanello) che segnali la richiesa di comunicazione è necessario predisporre un hardware più complicato. Prima di tutto occorrono una o più linee della CPU dedicate al ricevimento dei segnali di interrupt; in più occorre che lo stesso interrupt si faccia riconoscere dalla CPU, e anche questo rende più complessi i collegamenti; infine, anche se non sempre, è necessaria della circuiteria che consenta il servizio dei diversi dispositivi, in base alla priorità prefissata.
Tuttavia, dal momento che la tecnica dell'interrupt è così vantaggiosa dal punto di vista pratico, quasi tutti i costruttori di computer - escludendo casi particolari - vi ricorrono abitualmente. La possibilità di utilizzo degli interrupt è inoltre estesa anche ai normali programmatori da particolari istruzioni di cui è dotata la CPU. È quindi importante sapere dell'esistenza degli interrupt per due motivi: 1) chiunque li può usare nei propri programmi assembler; 2) il loro funzionamento aiuta a capire tutte quelle azioni che il computer esegue senza che vi sia un intervento diretto della mano dell'uomo.

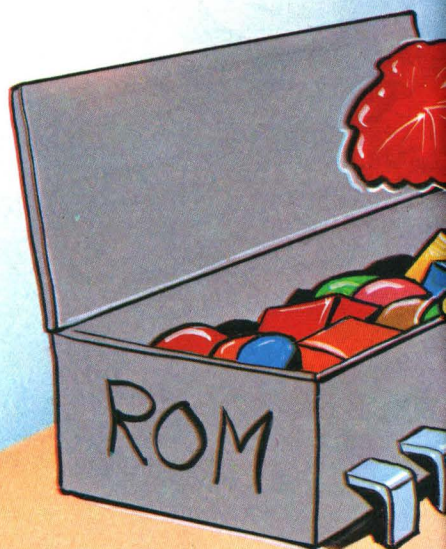
Usare la ROM

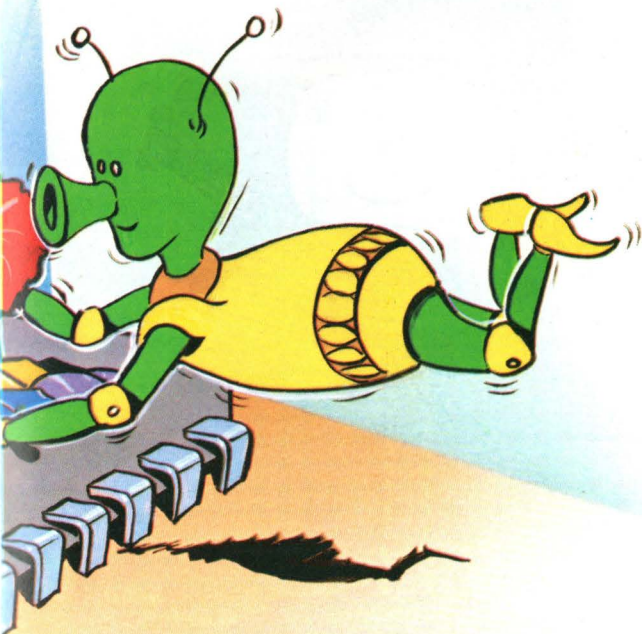
Esistono molte routine del sistema che possono consentire al programmatore esperto di risparmiare il tempo e la fatica di doverle riscrivere. I progettisti di calcolatori strutturano infatti le routine di sistema in maniera tale da poterle considerare come dei normali sottoprogrammi, richiamabili in qualsiasi momento sia da BASIC che da Assembler. Il vantaggio di una simile soluzione è più che evidente: si evita ai programmatori di dover affrontare tutte le volte gli stessi problemi (per esempio la visualizzazione dei risultati), consentendo loro di concentrarsi sul problema specifico piuttosto che sul problema generale. Inoltre le routine di sistema - poste nella memoria ROM e quindi non cancellabili - sono scritte e controllate da programmatori professionisti, con la conseguente garanzia di sicuro funzionamento. Per poter utilizzare una qualsiasi di queste routine l'unica cosa che si deve conoscere è l'indirizzo di partenza, oltre naturalmente ai registri del microprocessore che vengono interessati dalla routine stessa. Esistono diversi manuali che descrivono con notevole precisione tutte queste routine; noi tratteremo le principali, spiegando quindi (ed è questo che conta) come utilizzarle nei programmi. Ciascuna di queste routine possiede un particolare nome mnemonico - normalmente assegnato dalla casa madre - che le permette di essere distinta in modo semplice ed immediato dalle altre. La seguente tabella contiene quindi, oltre all'indirizzo di partenza, anche il nome di ciascuna routine.

NOME	INDIRIZZO	SCOPO
OUTDO	0018	scrive il contenuto dell'accumulatore nel dispositivo corrente.
CHKRAM	0000	routine di start comprendente, tra gli altri, il controllo della RAM.
LPTOUT	00A5	invia un carattere alla stampante.
BEEP	0C00	provoca un BEEP.
CLS	00C3	ripulisce lo schermo, anche quello grafico.
POSIT	00C6	muove il cursore testo nella colonna specificata in H e nella riga in L.
ERAFNK	00CC	cancella i messaggi dei tasti funzione.
DSPFNK	00CF	stampa i messaggi dei tasti funzione.
CHGCAP	0132	controlla lo stato della luce di CAPS LOCK. Accumulatore <> 0 = luce accesa.
TAPION	00E1	attiva il motore del registratore.
TAPIOF	00E7	arresta il motore del registratore.
CHGET	009F	preleva un carattere dal BUFFER della tastiera.

Come esempio di utilizzo di una di queste routine, vediamo in che modo è possibile stampare qualcosa sullo schermo. Dalla tabella appena scritta si deduce che la routine adibita alla visualizzazione dei caratteri sul video è OUTDO; essa richiede semplicemente che, prima della chiamata, il codice ASCII del carattere da visualizzare venga posto nell'accumulatore.

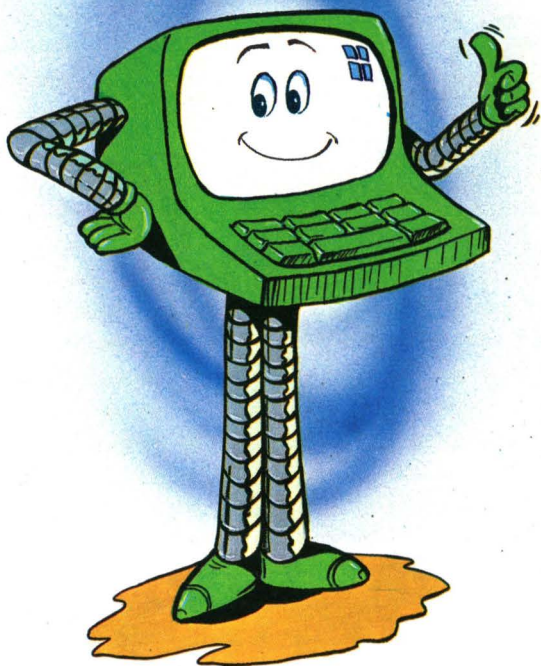
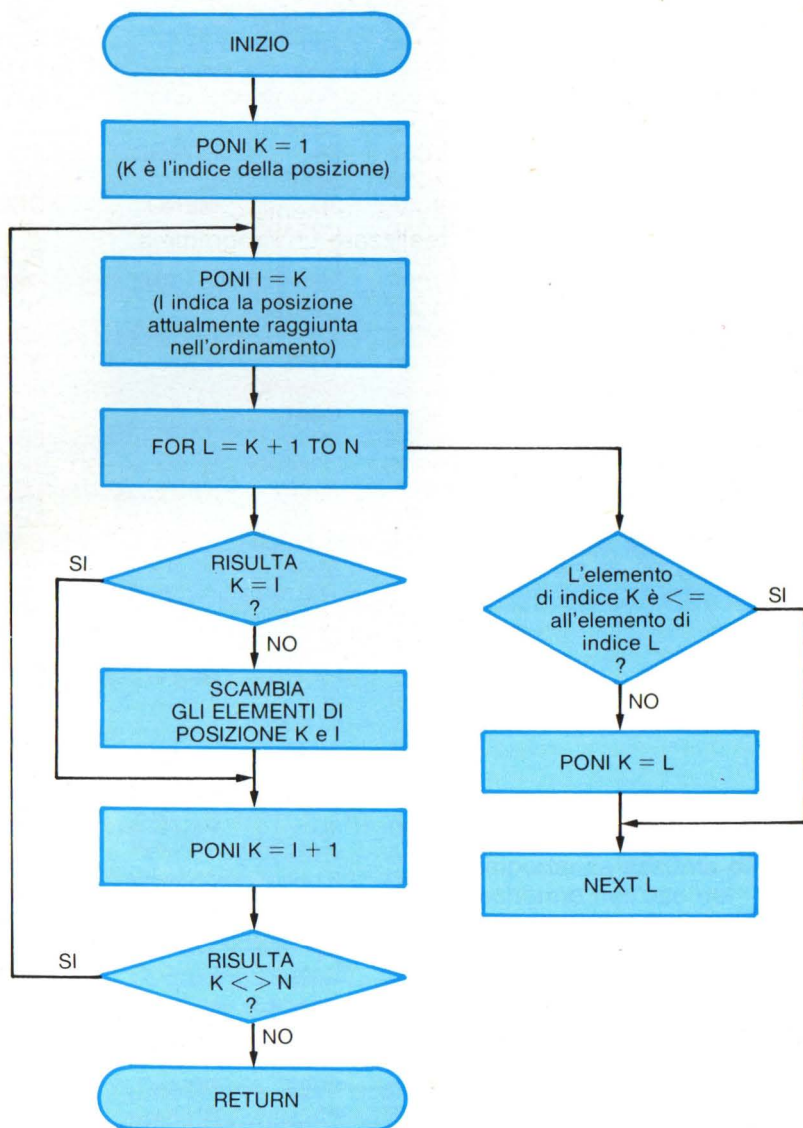
Dovremo quindi memorizzare un certo codice nell'accumulatore e chiamare la routine tante volte quanti saranno i caratteri che vogliamo far stampare. Per stampare la parola "ciao", potremo quindi scrivere:

```
LD A,43H     ;ASCII di "C"
CALL 0018

LD A,49H     ;ASCII di "I"
CALL 0018

LD A,41H     ;ASCII di "A"
CALL 0018

LD A,4FH     ;ASCII di "O"
CALL 0018

CALL 0C00
RET
```

All'operazione di stampa abbiamo anche aggiunto, al termine, una suonatina dell'altoparlante, mediante il ricorso alla subroutine BEEPER. Come puoi vedere, il fatto di conoscere l'esistenza di OUTDO ci ha evitato qualsiasi preoccupazione per quanto riguarda l'uscita dei risultati sul video. La chiamata avviene semplicemente attraverso la solita istruzione CALL, a ulteriore dimostrazione che nella ROM le routine si trovano scritte in forma di sottoprogrammi.

Ecco il listato Basic caricatore e, nella linea data, i codici esadecimali del programma in linguaggio macchina.

```
 10 CLEAR 200, 39999! : RESTORE : IN=40000!
 20 READ A$
 30 N=VAL("&H" +A$) : IF N>255 THEN 50 ELSE POKE IN,N
 40 IN=IN+1 : GOTO20
 50 DEFUSR=40000!
 60 A=USR(0)
 70 END
100 DATA 3E, 43, CD, 18, 00, 3E, 49, CD, 18, 00, 3E, 41, CD, 18, 00, 3E, 4F, CD, 18, 00,
    CD, 00, 0C, C9, FFF
```

PROGRAMMAZIONE

Bomba in linguaggio macchina

Alcuni suoni hanno bisogno della rapidità di esecuzione del linguaggio macchina per simulare realisticamente degli effetti speciali.
È il caso della bomba: quando è sganciata produce il caratteristico sibilo, sempre più intenso, fino all'esplosione finale.
Se hai in mente di realizzare un programma in cui ti serva una esplosione davvero realistica, non devi far altro che aggiungervi questa routine.
Ogni chiamata all'inizio del linguaggio macchina ti restituirà l'effetto speciale.

Lo schema a blocchi della routine di ordinamento è il seguente:

```
                            ORG     40000
9C40    3E07                LD      A,7
9C42    1EFE                LD      E,254
9C44    CD9300              CALL    147
9C47    3E08                LD      A,8
9C49    1E0F                LD      E,15
9C4B    CD9300              CALL    147
9C4E    1E28                LD      E,40
9C50    3E00    CICLO:      LD      A,0
9C52    CD9300              CALL    147
9C55    3E0A                LD      A,10
9C57    F5      RITARDO:    PUSH    AF
9C58    3EFF                LD      A,255
9C5A    3D      MENO:       DEC     A
9C5B    C25A9C              JP      NZ,MENO
9C5E    F1                  POP     AF
9C5F    3D                  DEC     A
9C60    C2579C              JP      NZ,RITARDO
9C63    7B                  LD      A,E
9C64    D696                SUB     150
9C66    CA6F9C              JP      Z,NEXT
9C69    C697                ADD     A,151
9C6B    5F                  LD      E,A
9C6C    C3509C              JP      CICLO
9C6F    3E00    NEXT:       LD      A,0
9C71    1E00                LD      E,0
9C73    CD9300              CALL    147
9C76    3E07                LD      A,7
9C78    1EF7                LD      E,247
9C7A    CD9300              CALL    147
9C7D    3E00                LD      A,0
9C7F    F5      NEXT1:      PUSH    AF
9C80    5F                  LD      E,A
9C81    CD9300              CALL    147
9C84    3E32                LD      A,50
9C86    F5      PIPPO:      PUSH    AF
9C87    3EFF                LD      A,255
9C89    3D      PLUTO:      DEC     A
9C8A    C2899C              JP      NZ,PLUTO
9C8D    F1                  POP     AF
9C8E    3D                  DEC     A
9C8F    C2869C              JP      NZ,PIPPO
9C92    F1                  POP     AF
9C93    D61F                SUB     31
9C95    CA9D9C              JP      Z,RITA
9C98    C620                ADD     A,32
9C9A    C37F9C              JP      NEXT1
9C9D    3E64    RITA:       LD      A,100
9C9F    F5      RITA1:      PUSH    AF
```

```
9CA0   3EFF                 LD     A,255
9CA2   3D        RITA2:     DEC    A
9CA3   C2A29C               JP     NZ,RITA2
9CA6   F1                   POP    AF
9CA7   3D                   DEC    A
9CA8   C29F9C               JP     NZ,RITA1
9CAB   3E07                 LD     A,7
9CAD   1EFF                 LD     E,255
9CAF   CD9300               CALL   147
9CB2   C9                   RET
```

```
 10 CLEAR 200, 39999! : RESTORE : IN=40000!
 20 READ A$
 30 N=VAL("&H" + A$) : IF N>255 THEN 50 ELSE POKE IN,N
 40 IN=IN+1 : GOTO20
 50 DEFUSR=40000!
 60 A=USR(0)
 70 END
100 DATA 3E, 07, 1E, FE, CD, 93, 00, 3E, 08, 1E, 0F, CD, 93, 00, 1E, 28, 3E, 00, CD, 93, 00
110 DATA 3E, 0A, F5, 3E, FF, 3D, C2, 5A, 9C, F1, 3D, C2, 57, 9C, 7B, D6, 96, CA, 6F, 9C, C6, 97
120 DATA 5F, C3, 50, 9C, 3E, 00, 1E, 00, CD, 93, 00, 3E, 07, 1E, F7, CD, 93, 00, 3E, 00, F5
130 DATA 5F, CD, 93, 00, 3E, 32, F5, 3E, FF, 3D, C2, 89, 9C, F1, 3D, C2, 86, 9C, F1, D6, 1F
140 DATA CA, 9D, 9C, C6, 20, C3, 7F, 9C, 3E, 64, F5, 3E, FF, 3D, C2, A2, 9C, F1, 3D, C2, 9F, 9C
150 DATA 3E, 07, 1E, FF, CD, 93, 00, C9, FFF
```

Le routine in ROM della videoram

Il BIOS (Basic Input Operating System) contiene numerose chiamate a routine in ROM riguardanti il processore video e la memoria video.

È ovvio, d'altra parte, vista l'enorme importanza assunta dallo schermo nell'uso del computer.

Nome: FILVRM

Funzione: riempire una certa area con un dato particolare. Puoi per esempio usarla per scrivere nel 768 byte della mappa di schermo il valore 32 (spazio), ottenendo un effetto CLS.
Paricolarità: HL deve contenere l'indirizzo di inizio del blocco di videoram. La lunghezza del blocco va in BC, il dato nell'accumulatore.

CALL: 0056H

Nome: LDIRMV

Funzione: ricopiare un blocco di memoria video nella RAM utente.
Particolarità: HL deve contenere l'indirizzo di inizio del blocco da copiare, BC la lunghezza e DE l'indirizzo da dove ricopiare nella RAM utente.

CALL: 0059H

Nome: LDIRVM

Funzione: ricopiare in videoram un blocco della memoria utente.
Particolarità: HL deve contenere l'indirizzo di inizio del blocco in memoria RAM, BC la lunghezza, DE il puntatore all'inizio nella videoram.

CALL: 005CH

Nome: WRTVRM

Funzione: questa routine in linguaggio macchina è l'equivalente dell'istruzione VPOKE del BASIC.
Particolarità: HL deve contenere l'indirizzo della videoram in cui scrivere; il dato (numero tra 0 e 255) va nell'accumulatore.

CALL: 004D

Nome: RDVRM

Funzione: è l'equivalente dell'istruzione BASIC VPEEK.
Particolarità: dopo la chiamata di questa routine, il registro A conterrà il valore letto nella videoram all'indirizzo puntato da HL.

CALL: 004A

Giracaratteri

È un programma in LM che utilizza due delle routine appena viste; esattamente la RDVRM e la WRTVRM.
Far ruotare l'intero set di caratteri in BASIC sarebbe una operazione lunghissima. In LM, invece, solo qualche istante ed il gioco è fatto. Ogni volta che il programma è chiamato, l'intero set dei caratteri viene copiato e riscritto, ruotato però di 90 gradi. Così, successivamente chiamate possono far eseguire ai caratteri il giro completo.
Non resta che scegliere la posizione più suggestiva o adattta agli scopi particolari del momento.

```
                                ORG     54000
D2F0    210000                  LD      HL,0
D2F3    3EFF                    LD      A,255
D2F5    F5          VIA:        PUSH    AF
D2F6    E5                      PUSH    HL
D2F7    3E08                    LD      A,8
D2F9    F5          DAI:        PUSH    AF
D2FA    CD18D3                  CALL    ROUTINE
D2FD    F1                      POP     AF
D2FE    23                      INC     HL
D2FF    SD                      DEC     A
D300    C2F9D2                  JP      NZ,DAI
D303    E1                      POP     HL
D304    E5                      PUSH    HL
D305    13                      INC     DE
D306    EB                      EX      DE,HL
D307    010800                  LD      BC,8
D30A    CD5C00                  CALL    92
D30D    E1                      POP     HL
D30E    010800                  LD      BC,8
D311    09                      ADD     HL,BC
D312    F1                      POP     AF
D313    3D                      DEC     A
D314    C2F5D2                  JP      NZ,VIA
D317    C9                      RET
D318    1162F2      ROUTINE:    LD      DE,62050
D31B    CD4A00                  CALL    74
D31E    47                      LD      B,A
D31F    3E08                    LD      A,8
D321    F5          LOOP:       PUSH    AF
D322    CD2FD3                  CALL    RUOTA
D325    F1                      POP     AF
D326    3D                      DEC     A
D327    C221D3                  JP      NZ,LOOP
D32A    C9                      RET
D32B    CD4A00                  CALL    74
D32E    47                      LD      B,A
D32F    1A          RUOTA:      LD      A,(DE)
D330    CB10                    RL      B
D332    CB17                    RL      A
D334    12                      LD      (DE)
D335    1B                      DEC     DE
D336    C9                      RET
```

```
10 CLEAR 200, 53999! : RESTORE : IN=54000!
20 READ A$
30 N=VAL("&H"+A$) : IF N>255 THEN 50 ELSE POKE IN,N
40 IN=IN+1 : GOTO20
50 DEFUSR = 54000! : SCREEN 1
60 A=USR(0)
70 END
100 DATA 21, 00, 00, 3E, FF, F5, E5, 3E, 08, F5, CD, 18, D3, 23, 3D
110 DATA C2, F9, D2, E1, E5, 13, EB, 01, 08, 00, CD, 5C, 00, E1, 01, 08
120 DATA 00, 09, F1, 3D, C2, F5, D2, C9, 11, 62, F2, CD, 4A, 00, 47, 3E, 08, F5, CD, 2F, D3, F1, 3D,
C2, 21, D3, C9, CD, 4A, 00, 47, 1A
130 DATA CB, 10, CB, 17, 12, 1B, C9, FFF
```

VIDEOESERCIZI

Quali sono i più veloci tra i due seguenti programmi?
Cronometro alla mano cerca di scoprirlo e di spiegartene il motivo. Ricorda però che la velocità può essere importante, a volte essenziale, ma che la leggibilità e la chiarezza del listato lo sono ancora di più.

A
```
10 FOR P = 1 TO 1000
20 REM Questo commento rallenta; a meno che non
     sia indispensabile è meglio ometterlo
30 NEXT P
```

B
```
10 FOR P = 1 TO
     1000 : NEXT P
```

A
B

A
```
10 CLS
20 LET N$ = "23"
30 FOR I = 1 TO 1000
40 PRINT N$;
50 NEXT I
```

B
```
10 CLS
20 LET N = 23
30 FOR I = 1 TO 1000
40 PRINT N;
50 NEXT I
```

A
B